J. Korean Soc. Ind. Appl. Math. Vol.27, No.4, 324–341, 2023 https://doi.org/10.12941/jksiam.2023.27.324

A NODE PREDICTION ALGORITHM WITH THE MAPPER METHOD BASED ON DBSCAN AND GIOTTO-TDA

DONGJIN LEE[1] AND JAE-HUN JUNG[2,†]

[1]GRADUATE SCHOOL OF ARTIFICIAL INTELLIGENCE, POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY, 37673 SOUTH KOREA
Email address: dongjinlee@postech.ac.kr

[2]DEPARTMENT OF MATHEMATICS, POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY, 37673 SOUTH KOREA
Email address: †jung153@postech.ac.kr

ABSTRACT. Topological data analysis (TDA) is a data analysis technique, recently developed, that investigates the overall shape of a given dataset. The mapper algorithm is a TDA method that considers the connectivity of the given data and converts the data into a *mapper* graph. Compared to persistent homology, another popular TDA tool, that mainly focuses on the homological structure of the given data, the mapper algorithm is more of a visualization method that represents the given data as a graph in a lower dimension. As it visualizes the overall data connectivity, it could be used as a prediction method that visualizes the new input points on the mapper graph. The existing mapper packages such as Giotto-TDA, Gudhi and Kepler Mapper provide the *descriptive* mapper algorithm, that is, the final output of those packages is mainly the mapper graph. In this paper, we develop a simple *predictive* algorithm. That is, the proposed algorithm identifies the node information within the established mapper graph associated with the new emerging data point. By checking the feature of the detected nodes, such as the anomality of the identified nodes, we can determine the feature of the new input data point. As an example, we employ the fraud credit card transaction data and provide an example that shows how the developed algorithm can be used as a node prediction method.

1. INTRODUCTION

Topological data analysis (TDA) is one of the most popularly studied data analysis techniques these days. Unlike traditional statistical data analysis, TDA focuses more on the *shape* of data. By examining the shape of data, important data characteristics, possibly overlooked through traditional analysis, could be revealed [1].

Received November 30 2023; Revised December 19 2023; Accepted in revised form December 23 2023; Published online December 25 2023.

2000 *Mathematics Subject Classification.* 93B05.

Key words and phrases. Topological data analysis, Mapper algorithm, Prediction algorithm, Fraud transaction.

† Corresponding author.

Persistent homology is a TDA technique that considers the homological structure of data such as cyclic structure of the given data and visualizes the result as a diagram, e.g. persistence barcode, persistence diagram, etc. Research of combining TDA and machine learning centers around the usages of these barcodes and diagrams in the machine learning workflow [2]. The typical procedure of persistent homology is to take the input data as a point cloud and build the complex out of it via the so-called filtration procedure. Filtration procedure involves the computation of homology at different scales and the final visualization is the collection of the changes of homology through scales. Since the early works of persistent homology [3, 4], various advanced methods have been developed and applied to various applications. The key element of this method is to take the given data as a geometrical object, for which the given data is transformed into a point cloud. Using the constructed point cloud the associated complex is being built to imitate the original data. During the imitation procedure, important data characteristics, e.g. homology, are computed. It is not straightforward to transform the given data into a point cloud. For example, for time-series data, the sliding window embedding method is a popular method to construct a point cloud out of the time-series data [5, 6]. Recently, a direct method is also proposed that does not rely on the sliding window embedding but instead uses the orthogonality of the Fourier bases on N-torus [7]. In [8], a 3D spherical projection is proposed to efficiently project the 3D vascular flows as a point cloud suitable to TDA analysis. However, there is no unified theory to determine the best transform towards the point cloud. Rather, it highly depends on the given data. For this reason, it is important to understand well the given data before applying TDA.

The mapper [9] algorithm is also a TDA technique that considers the shape of data. Unlike persistent homology, explained briefly above, the mapper algorithm is more of a visualization method. As explained above, the final visualization of persistent homology, e.g. persistence diagram, is plugged into a machine learning workflow. However, the mapper visualization in a machine learning context is less popular at the moment compared to persistent homology. The mapper algorithm visualizes the given data as a *mapper* graph. That is, the mapper algorithm defines the vertices and edges out of the data and constructs the graph. The original mapper idea was proposed in [9] and there have been various developments since then. The main difference between persistent homology and the mapper algorithm is that persistent homology focuses more on the homology of the shape of data but the mapper algorithm focuses more on the shape of the given data. For example, it is possible that the mapper algorithm could distinguish those two homologically equivalent objects while those two datasets are categorized into the same class with the persistent homology approach. In that sense, the mapper algorithm provides more intuitive visualization of the given data towards interpretation. Here we note that for both persistent homology and the mapper method it is necessary to properly transform the given data into a point cloud. As stated in the above, though, the optimal transform may depend on data.

In this work, we use the visualization characteristics of the mapper algorithm to develop a node prediction algorithm. That is, we develop an algorithm that first identifies the nodes associated with the input data point within the considered mapper graph and then visualizes the individual input data in the mapper graph. Popular mapper toolkits such as Giotto-TDA [10] do

not provide the algorithm that maps the individual new input data into the already established mapper graph domain. By utilizing this visualization, it is efficient to identify the connectivity of the new data point within the overall mapper graph, which also makes it possible to identify the features of the input data visually. The computational modules of the developed prediction algorithm are available at `https://github.com/HiddenBeginner/mapper_prediction/`.

As a numerical example, we apply the prediction algorithm to the credit card fraud transaction data and show how the prediction algorithm identifies the associated nodes with the input data. For this example, we mainly focus on the supervised algorithm. That is, for the visualization, we assume that we already have the knowledge about which points indicate the fraud data.

This paper is composed of the following sections. In Section 2, we will provide the overall description of the mapper algorithm. In this section, we use two well-known examples to help the understanding. In Section 3, we propose a node prediction algorithm based on Giotto-TDA and DBSCAN. In Section 4, we provide an application example of the developed prediction algorithm using the credit card fraud transaction data. In Section 5, we provide a brief conclusion. In Appendix A, we provide a partial code of the prediction algorithm that draws circles around the identified nodes for the input data. In Appendix B, feature distributions of the credit card transaction data are provided based on which the projection dimensions are determined for the numerical experiments.

2. MAPPER ALGORITHM

2.1. **Mapper algorithm.** The mapper algorithm is a TDA method that visualizes the given data defined in a high-dimensional space, in the form of a graph in a low-dimensional space. There are various data visualization methods such as PCA (principle component analysis) [11] and t-SNE (t-distributed stochastic neighbor embedding) [12] methods. These methods are all dimensional reduction algorithms for high-dimensional datasets. Unlike these dimensional reduction algorithms, the mapper algorithm uses the concept of graph and focuses on the connectivity of the given data points. Below we explain how the mapper algorithm constructs the graph out of the given data.

Suppose that a dataset is given and it is represented as a point cloud X and that X is composed of N distinct data points, i.e. $X = \{\mathbf{x}_i \mid i = 1, \dots, N\} \subset \mathcal{X}$ where $\mathcal{X}$ is the underlying topological space in which X is generated. Each $\mathbf{x}_i$ could be a vector. As stated in the Introduction, the method of converting the provided data into a suitable point cloud X is not unique. Indeed, identifying the ideal transform for the provided dataset is not a simple task; it demands additional exploration of the data. The construction of a suitable point cloud for the given data holds crucial importance in the success of TDA. For instance, in [8], vascular data was projected onto a sphere, proving highly effective for TDA of vascular flow. Similarly, the utilization of sliding window embedding for time-series data [5, 6] is another exemplary approach. One direct approach involves employing the identity map. The specifics of the mapper algorithm may vary depending on the problems under consideration. However, in this paper,

we adopt the conventional mapper algorithm [1]. For the mapper algorithm, numerous free parameters need to be predefined, and the performance varies based on the selected parameter values. A unified theory regarding the best parameter values for optimal performance is not available. There are several approaches for parameter optimization [13].

Filter function: The mapper algorithm first defines the so-called lens or filter function $f : X \to Y$, where f is a continuous map and Y is typically defined in a dimension lower than that of X. The filter function f is either given or to be defined depending on the problem. Suppose that X and f are given, i.e. we have the sets of $\{\mathbf{x}_i\}$ and $\{f(\mathbf{x}_i)\}$. If $Y \subset \mathbb{R}$, it is convenient to define the minimum and maximum of $f(\mathbf{x}_i)$,

$$m := \min_{i=1,\cdots,N} f(\mathbf{x}_i),$$

and

$$M := \max_{i=1,\cdots,N} f(\mathbf{x}_i).$$

Then, $m \leq f(\mathbf{x}_i) \leq M$ for $i = 1, \cdots, N$. The values of m and M are used to determine the size of the cover explained below.

Covering: Suppose that Y is equipped with a covering $\mathcal{U} = \{U_i \mid i = 1, 2, \cdots, r < \infty\}, Y \subset \cup_i U_i$. Here the adjacent U_i are supposed to overlap. Let r be the parameter that determines the number of covers and p be the degree of overlapping as a probability such that $0 < p < 1$. Practically p is chosen as $0 < p \leq \frac{1}{2}$. The choice of U_i is not unique. One can use a uniform covering, that is, U_i are distributed uniformly with the same degree of overlap for any two neighboring covers. For example, consider the case of $Y \subset \mathbb{R}$ and $\cup_i U_i = (0, 2)$. Choose $r = 4$ and $p = \frac{2}{3}$. Then each U_i is an interval $I_i \subset \mathcal{U}$ and the uniform covering is

$$\mathcal{U} = \{I_i\} = \left\{(0, 1), \left(\frac{1}{3}, \frac{4}{3}\right), \left(\frac{2}{3}, \frac{5}{3}\right), (1, 2)\right\}.$$

The d-dimensional extension of the one-dimensional uniform covering is called a *cubical cover*. In specific, a cubical cover $\mathcal{U}$ is a set of d-dimensional cubes:

$$\mathcal{U} = \left\{I_{k_1}^{(1)} \times \cdots \times I_{k_d}^{(d)} \mid k_1, \ldots, k_d = 1, \ldots, r\right\},$$

where r is the number of intervals of each dimension and $I_k^{(j)} = (a_k^{(j)}, b_k^{(j)})$ represents open interval along the jth dimension with $a_k^{(j)} < a_{k+1}^{(j)}, b_k^{(j)} < b_{k+1}^{(j)}, a_1^{(j)} = m^{(j)}$ and $b_r^{(j)} = M^{(j)}$ for $m^{(j)} = \min_i f_j(\mathbf{x}_i)$ and $M^{(j)} = \max_i f_j(\mathbf{x}_i)$. Here $f_j(\mathbf{x}_i)$ is the jth element of $f(\mathbf{x}_i)$. In practice, the intervals of each dimension are set to be the same length (i.e. $b_1^{(j)} - a_1^{(j)} = \cdots = b_r^{(j)} - a_r^{(j)}$). When identifying the node where the newly injected data point $\mathbf{x}^*$ belongs, we consider $f_j(\mathbf{x}^*)$ to be contained in $I_1^{(j)}$ if $f_j(\mathbf{x}^*) \leq m^{(j)}$. Similarly, $f_j(\mathbf{x}^*)$ is contained in $I_r^{(j)}$ if $f_j(\mathbf{x}^*) \geq M^{(j)}$. Two consecutive intervals, I_k and I_{k+1}, overlap with the ratio p of the length of the overlapped interval to the length of each interval, i.e. (i.e. $p = (b_k - a_{k+1})/(b_k - a_k)$ or $p = (b_k - a_{k+1})/(b_{k+1} - a_{k+1})$). Here note that the uniform cubical covering was utilized for the original mapper algorithm, but the covering can be adaptively determined, e.g. [13].

Inverse covering: Since we want to construct a graph where the vertices form as a cluster of $\mathbf{x}_i$, we find $f^{-1}(U_i)$. Since f is assumed to be continuous $f^{-1}(\mathcal{U})$ is also an open covering of X. For the one-dimensional case as in the above step, for example, $X_i := f^{-1}(I_i) = \{\mathbf{x} \mid f(\mathbf{x}) \in I_i, \mathbf{x} \in \mathcal{X}\}$. This step is crucial to form vertices towards the mapper graph. Once the inverse covering $X_i = f^{-1}(U_i)$ and all the points of $\mathbf{x}_i \in \{\mathbf{x} \mid \mathbf{x} \in X_i, \mathbf{x} \in X\}$ are identified, we now examine how the points $\mathbf{x}_i$ are connected to each other in X_i. This examination is done by counting the number of clusters within X_i, which is usually achieved by using the clustering algorithms. A clustering algorithm is applied to the set of points $\{\mathbf{x} \mid \mathbf{x} \in X \text{ and } \mathbf{x} \in X_i\}$. Decompose X_i into path-connected sets $X_{i,k}$ where k is finite, $k = 1, 2, \cdots, K$ and K is the total number of connected regions in X_i. The connected sets $X_{i,k}$ are determined by the clustering algorithm applied to the set $\{\mathbf{x} \mid \mathbf{x} \in X \text{ and } \mathbf{x} \in X_i\}$.

Vertices and edges: Now we treat $X_{i,k}$ as a vertex and whose size is determined by the number of $\mathbf{x} \in X$ in X_i for visualization. Usually, the color is assigned to each vertex and every vertex $X_{i,k}$ in X_i is assigned the same color for visualization. An edge is constructed between two vertices $X_{i,k}$ and $X_{j,l}$, $i \neq j$ if $U_i \cap U_j \neq \emptyset$ and there exists $\mathbf{x} \in X$ such that $\mathbf{x} \in X_{I,k}$ and $\mathbf{x} \in X_{j,l}$.

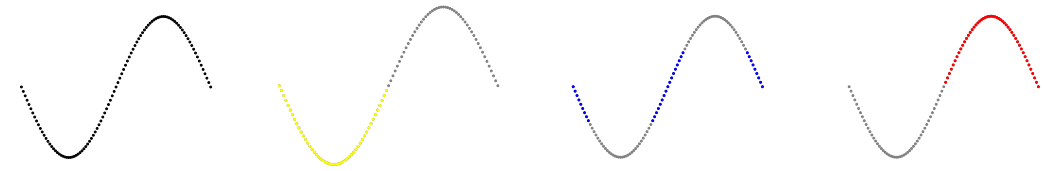
Remark: The mapper algorithm focuses on the connectivity of points $\mathbf{x} \in X$ according to the filter function values and the final result is given as a graph. This does not imply that there is a unique visualization of the graph. That is, the visualization components of the graph such as the edge lengths, the coordinates of the vertices, etc. are arbitrary.

2.2. **Clustering algorithms.** For the determination of the vertices and edges for the mapper algorithm, clustering algorithms are used. There are several clustering algorithms including DBSCAN (Density-based spatial clustering of applications with noise) [14]. In this work, we also adopt DBSCAN for clustering. In practice, DBSCAN algorithm is popularly used because it is not required to specify the number of clusters, which is the main difference between DBSCAN and k-means clustering algorithms. DBSCAN is also suitable for finding arbitrarily-shaped clusters and is robust to noise and outliers. The DBSCAN algorithm has two parameters: `eps`, which will be denoted as ϵ for conciseness, and `min_samples` for denseness. For each point $\mathbf{x}_i$ in a given dataset $\mathcal{D}$, define its ϵ-neighborhood $B_\epsilon(\mathbf{x}_i)$ as the set of all points in $\mathcal{D}$ that are at distance less than ϵ:

$$B_\epsilon(\mathbf{x}_i) := \{\mathbf{x} \in \mathcal{D} \mid d(\mathbf{x}_i, \mathbf{x}) < \epsilon\}.$$

If the ϵ-neighborhood $B_\epsilon(\mathbf{x}_i)$ of a point $\mathbf{x}_i$ contains at least `min_samples` points, the point $\mathbf{x}_i$ is considered as a *core point* and its ϵ-neighborhood $B_\epsilon(\mathbf{x}_i)$ forms a cluster. If the ϵ-neighborhood $B_\epsilon(\mathbf{x}_i)$ includes another core point $\mathbf{x}_j$, the two clusters formed by these ϵ-neighborhoods merge into a single cluster. If a point is not a core point but belongs to some cluster, it is called a *border point*. If a point is not part of any cluster, it is regarded as *noise*. For an unseen data point $\mathbf{x}^*$, if there are core points in $B_\epsilon(\mathbf{x}^*)$ it is assigned to the cluster that includes the closest core point. If there is no core point in $B_\epsilon(\mathbf{x}^*)$, it is classified as noise.

2.3. **Examples.** In this section, we provide two examples. These examples are simple enough to draw the final mapper graph by hand. We provide the sample codes using Giotto-TDA.



(A) $f(x) = \sin(\pi x)$ (B) $f^{-1}\big([-1, 0)\big)$ (C) $f^{-1}\big((-0.5, 0.5)\big)$ (D) $f^{-1}\big((0, 1]\big)$

FIGURE 1. The filtering function $f(x)$ and the covering of $\mathcal{U}$. Those points that belong to each covering are colored with yellow, blue, and red colors.

Example 1: Consider a set of evenly spaced numbers from -1 to 1:

$$X = \{x_i \mid x_i = x_0 + n\Delta x,\ n = 0, 1, \ldots, 1000\}, \quad x_0 = -1, \quad \Delta x = 0.002.$$

We use a sine function for $f(x) = \sin \pi x$ as the filter function. Then, the range of f is $[-1, 1]$. We choose the uniform covering with the resolution $r = 3$ and the overlapping degree $p = 0.5$. Then for the uniform covering with $r = 3$ and $p = 0.5$, we have

$$\mathcal{U} = \{I_i\} = \{[-1, 0), (-0.5, 0.5), (0, 1]\}.$$

Figure 1 shows the image of x_i, i.e., $f(x_i)$ in each cover. As shown in the figure, the yellow, blue and red colors are assigned to the coverings, respectively. Note that the points x_i are evenly spaced, whereas the corresponding values $f(x_i)$ on the curve of $f(x)$ are not uniformly distributed.

As shown in the figure, only the cover I_2 has the inverse image, $f^{-1}(I_2)$, that has the three connected sets of points. Other covers have a single connected set. That is, there are three nodes in X_2 corresponding to I_2 while each of X_1 and X_3 forms a single node. Figure 2 shows the final mapper graph for X where the node size is proportional to the number of points that belong to each $X_{i,k}$. The color of each node is the color in Fig. 1.

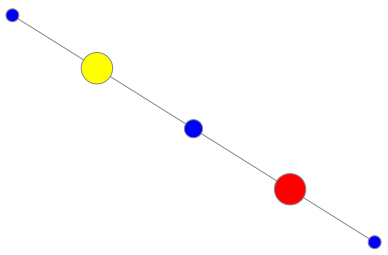


FIGURE 2. The Mapper graph for Example 1. $f(x) = \sin(\pi x)$

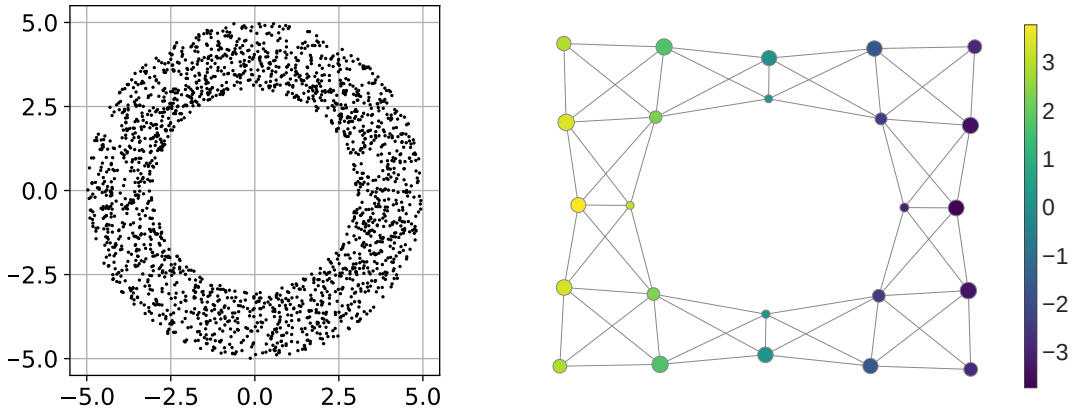


FIGURE 3. Left: The annulus data in 2D. Right: The corresponding mapper graph via Giotto-TDA.

Example 2: In this example, we consider the annulus data in 2D and use Giotto-TDA [10] to generate the mapper graph. Consider the annulus data given in the left figure of Fig. 3. The inner and outer radii are 3 and 5, respectively.

For this case, we use two filter functions $f_1(x, y)$ and $f_2(x, y)$. For simplicity, we choose

$$f_1(x, y) = x, \quad f_2(x, y) = y.$$

For the 2D cover, we use the rectangle for each cover. In Giotto-TDA there is a module *CubicalCover* for handling this type of cover. For the clustering, we use DBSCAN algorithm. We also use $r = 5$ and $p = 0.5$. The following shows the Giotto-TDA script for this problem:

```
# Data X
X = annulus(C=[0,0],ir=3,er=5)
# 1. Filter function
filter_func = Projection(columns=[0, 1])
# 2. Cover
cover = CubicalCover(n_intervals=5, overlap_frac=0.5)
# 3. Clustering
clusterer = DBSCAN()
# 4. Mapper algorithm
pipe = make_mapper_pipeline(filter_func=filter_func,
        cover=cover,
        clusterer=clusterer,
        verbose=False)
fig = plot_static_mapper_graph(pipe, X)
```

Algorithm 1 Mapper inference

Input: Target data $\mathbf{x}^*$
Output: Set $\mathcal{V}^*$ of vertices that include $\mathbf{x}^*$

Initialize a vertex set $\mathcal{V}^* = \emptyset$
Evaluate the filter function value $f(\mathbf{x}^*)$
Find a set $\mathcal{U}^*$ of covers containing $f(\mathbf{x}^*)$ (i.e. $\mathcal{U}^* = \{U_i \mid f(\mathbf{x}^*) \in U_i\}$)
if $\mathcal{U}^* = \emptyset$ **then**
  return $\mathcal{V}^*$
else
  for Cover U_i in $\mathcal{U}^*$ **do**
    Find the inverse covering $X_i = f^{-1}(U_i)$
    if $\exists$ a cluster $X_{i,j}$ that $\mathbf{x}^*$ belongs to **then**
      $\mathcal{V}^* \leftarrow \mathcal{V}^* \cup \{X_{i,j}\}$
    end if
  end for
  return $\mathcal{V}^*$
end if

```
fig.show()
```

The right figure of Fig. 3 shows the Mapper graph via Giotto-TDA. As shown in the figure, the mapper graph does not necessarily look similar to the given annulus, but it preserves the connectivity of the data in the graph.

3. Prediction Algorithm

As described in the previous section, for the given dataset $X = \{\mathbf{x}_i \mid i = 1, \cdots, N\}$, the mapper algorithm generates the mapper graph from X. This graph, corresponding to X, based on specified parameters such as r and p, illustrates the overall shape of X. Should new data be introduced, such as an additional point, and if we can locate the specific nodes within the mapper graph where this new data belongs, we can utilize the mapper algorithm as a real-time prediction algorithm for emerging data. Essentially, this represents the predictive capacity of the mapper algorithm.

The existing mapper software packages such as Giotto-TDA [10], Gudhi [15], and Kepler Mapper [16], do not support this prediction function mainly due to their reliance on clustering algorithms implemented within Scikit-learn [17]. For example, the following modules provided in Scikit-learn, which are widely used as clustering algorithms for the construction of the mapper graph are *descriptive* algorithms but not *predictive*:

- `DBSCAN`
- `AgglomerativeClustering`

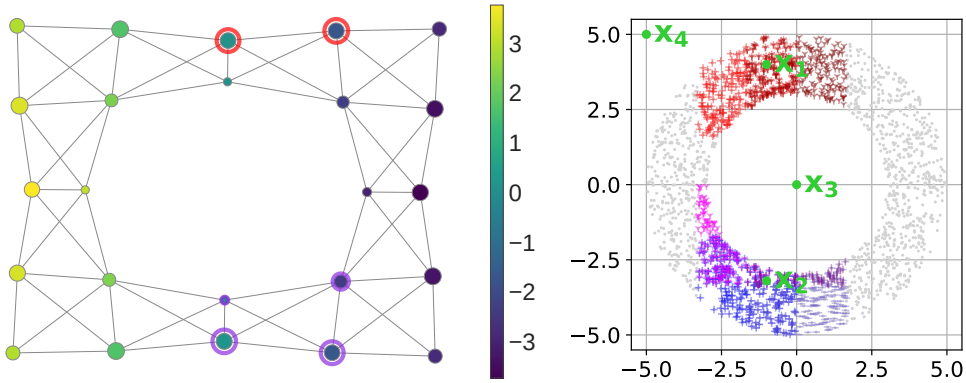
However, if such a predictive capability of the mapper algorithm can be combined with the existing packages, it could offer an efficient prediction method without perturbing the already constructed mapper graph. In this paper, we develop a simple predictive algorithm that provides such a predictive method, by identifying the nodes associated with the input data, compatible with the existing mapper packages. For the implementation, we build the method based on `DBSCAN` from Scikit-learn and the mapper algorithm available from Giotto-TDA.

The predictive algorithm is simply given as the following: Suppose that we are given a point $\mathbf{x}^*$ that is possibly in X and is not necessarily used to construct the mapper graph. Our objective is to identify the specific nodes within the constructed mapper graph where the input point $\mathbf{x}^*$ belongs. If these nodes fall within an anomaly sub-graph, the likelihood of $\mathbf{x}^*$ being anomalous is significantly elevated. Note that there could be multiple such nodes for $\mathbf{x}^*$. By applying the filter function f to $\mathbf{x}^*$, we first determine the covers, U_i associated with the filter value $f(\mathbf{x}^*)$. Also note that there could be multiple covers that contain $f(\mathbf{x}^*)$. Let $\mathcal{U}^*$ be the set of such covers, $\mathcal{U}^* = \{U_i \mid f(\mathbf{x}^*) \in U_i\}$. Once the cover set $\mathcal{U}^*$ is determined, the pre-image of $f^{-1}(U_i)$ are computed and the corresponding X_i are determined. Once X_i are determined, the cluster component within each X_i that includes $\mathbf{x}^*$ is identified. As such a cluster is represented as a known node in the mapper graph, our prediction algorithm returns such node information. Again note that there could be multiple nodes that satisfy the given condition.

The pseudo-algorithm for the prediction method that returns the node information $\mathcal{V}^*$ to which the given data $\mathbf{x}^*$ belongs within the already existing mapper graph is presented in Algorithm 1. In the algorithm, the input is a point $\mathbf{x}^*$ for which we want to determine the corresponding nodes in the mapper graph, and the output of the algorithm is the node set $\mathcal{V}^*$ that contains $\mathbf{x}^*$.

Example 3: In this example, we use the annulus data considered in Example 2 and show how the prediction algorithm identifies the node information for the input data points. Figure 4 illustrates the prediction results for unseen points $\mathbf{x}_1 = (-1, 4)$, $\mathbf{x}_2 = (-1, -3.2)$, $\mathbf{x}_3 = (0, 0)$ and $\mathbf{x}_4 = (-5, 5)$ on the mapper graph in Example 2. Note that the filter function in Example 2 is the identity function. The point $\mathbf{x}_1$ is included in the two overlapping covers and $\mathbf{x}_2$ is assigned to four inverse coverings. The points $\mathbf{x}_3$ and $\mathbf{x}_4$ are part of the inverse coverings, which contain no points. Figure 4 (A) shows the mapper graph of the annulus data with the assigned nodes of $\mathbf{x}_1$ and $\mathbf{x}_2$, highlighted in red and violet circles, respectively. For each of $\mathbf{x}_1$ and $\mathbf{x}_2$, the assigned nodes are fully connected since each pair of inverse coverings has the nonempty intersection, which is depicted in Fig. 4 (B). The script below demonstrates how to give the input data for unseen points and retrieve the list of nodes to which these points belong. The returned value of -1 indicates that there is no matching node for the input data point. For example, the points $(x, y) = (0, 0)$ and $(-5, 5)$ do not correspond to any nodes in the mapper graph. The return value for $(x, y) = (-1, 4)$ is $[7, 10]$, which indicates that there are two matching nodes (node numbers 7 and 10) to $(x, y) = (-1, 4)$.[1]

[1] `pmapper` is our custom module, available at `https://github.com/HiddenBeginner/mapper_prediction`



(A) (B)

FIGURE 4. Prediction for unseen points $\mathbf{x}_1 = (-1, 4)$, $\mathbf{x}_2 = (-1, -3.2)$, $\mathbf{x}_3 = (0, 0)$ and $\mathbf{x}_4 = (-5, 5)$. (A): The assigned nodes in the mapper graph of $\mathbf{x}_1$ (marked as red circles) and $\mathbf{x}_2$ (marked as violet circles). (B): The points in the inverse coverings corresponding to the four points (marked as green circles). Different colors and symbols represent distinct inverse coverings.

```
# Preditive algorithm
# Input: data points
# Output: node information

import numpy as np
from pmapper.utils import data2nodes

data = np.array([
    [-1.0, 4.0],
    [-1.0, -3.2],
    [0.0, 0.0],
    [-5.0, 5.0] ])
nodelist = data2nodes(data, pipe, graph)
print(nodelist)
[[7, 10], [16, 2, 17, 22], -1, -1]
```

4. EXPERIMENT

For the numerical experiment, we use the anomaly detection problem and we assume that the anomaly points within X are represented as a distinctive set of nodes in X. We focus on how to identify the nodes where the new input points belong. If the identified nodes are

anomaly nodes, we expect that the new input point is also anomalous. To demonstrate the developed prediction algorithm, we consider a more realistic example, i.e. the credit card fraud transaction data explained below. Here note that it is not the goal of this paper to develop an anomaly detection method. Our goal is to develop a node prediction algorithm. We assume that once we identify the associated nodes with the input data we can know the features of the input data by examining the features of the identified nodes.

4.1. **Credit card fraud transaction data.** To use a realistic problem for the experiment, we consider the anomaly detection problem of the credit card fraud data provided by Kaggle.[2] The data is composed of total $284,807$ individual transactions. The individual data is composed of 31 elements with the first denoting the time stamp and the second last the total amount of transaction. The last element has the value of either 0 or 1. The value of 1 indicates that the transaction is the fraud transaction. The value of 0 indicates the normal transaction. The meaning of the rest 28 elements, named as `V1` to `V28`, is not given. Since we know whether the data is fraud or not, this problem is for the fraud detection with supervised data. Among all, there are 492 fraud data, which is about 0.1727% of the data. We used the first 20% of data to construct mapper graphs, primarily due to the heavy space complexity of our predictive mapper algorithm. The remaining data were used for evaluation. We excluded the time stamp and the label when constructing mapper graphs.

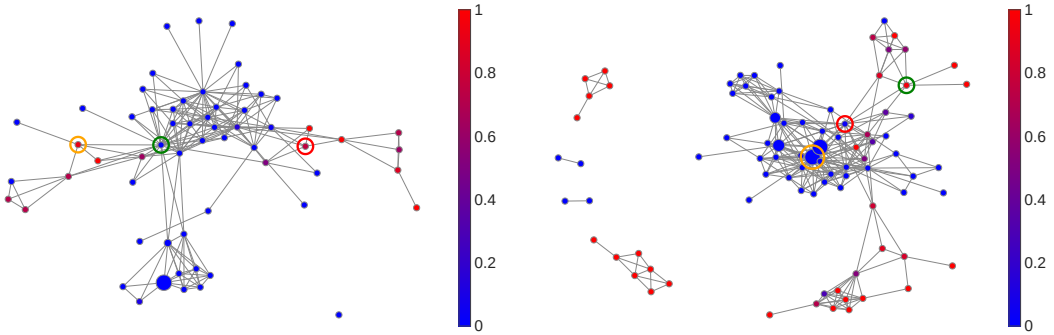
4.2. **Construction of mapper graph.** The filter function is designed to distinctly separate abnormal data in the codomain, drawing inspiration from the tutorial provided by Kepler Mapper.[3] Specifically, the filter function $f : X \to \mathbb{R}^3$ for $X \subseteq \mathbb{R}^{29}$ is defined by

$$f(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{bmatrix}, \tag{4.1}$$

where $f_1(\mathbf{x})$ is the anomaly score of $\mathbf{x}$ computed by the Isolation Forest [18] algorithm trained on X, $f_2(\mathbf{x})$ is the distance from its nearest neighborhood (excluding its own), and $f_3(\mathbf{x})$ is the Euclidean norm of $\mathbf{x}$. Since no function f_i requires label information (indicating whether each data is normal or fraudulent), the filter function f can be applied to an unlabeled dataset. However, one can define a filter function that effectively distinguishes normal and fraudulent data in the codomain by utilizing the label information. For example, the projection onto the k features that distinctly separate normal and abnormal data can serve as such a filter function. Building upon this intuition, we define another filter function g as the projection to the k features that exhibit clear segregation in the histograms between normal and abnormal data. In specific, the projection onto the $k = 3$ features `V4`, `V12` and `V14` is used as the filter function

[2]https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[3]https://kepler-mapper.scikit-tda.org/en/latest/notebooks/TOR-XGB-TDA.html



(A) Mapper graph $\mathcal{G}_f$ (B) Mapper graph $\mathcal{G}_g$

FIGURE 5. Mapper graphs with the two different filter functions f (left) and g (right). The color of each node indicates the proportion of abnormal data in the node. The colored circles indicate the top3 nodes where fraudulent data from the test dataset are most frequently mapped. The colors correspond to those used in Figure 6.

g:

$$g(\mathbf{x}) = \begin{bmatrix} v_4 \\ v_{12} \\ v_{14} \end{bmatrix}, \tag{4.2}$$

where v_4, v_{12} and v_{14} are the values of each feature element of `V4`, `V12` and `V14`, respectively. Appendix B shows the distribution of each feature from `V1` to `V28` for the normal (blue) and fraud data (orange) in the training dataset. As shown in the figure, there are clear differences between the normal and abnormal distributions for `V4`, `V12` and `V14`. There are other features that also show the distributional difference between the normal and abnormal data, such as `V9`, `V10`, `V16`, `V17`, `V18`, and `V19`. However, in this work, we focus on those three features of `V4`, `V12` and `V14`, which is sufficient for our main objective of our research.

We use a cubical cover with $r = 5$ and $p = 0.2$ for a covering of the range of each filter function. We employ the DBSCAN clustering algorithm with `eps`$= 0.5$ and `min_samples`$= 5$ to identify connected components in each of the inverse covering. All hyperparameters are selected by trial and error to ensure the generated mapper graphs are both visually interpretable and explainable.

4.3. **Results.** Figure 5 shows the two mapper graphs, denoted as $\mathcal{G}_f$ (left) and $\mathcal{G}_g$ (right), with the different filter functions f in (4.1) and g in (4.2), respectively. The size of each node is proportional to the number of data within the node while the color indicates the proportion of fraud data in the node.

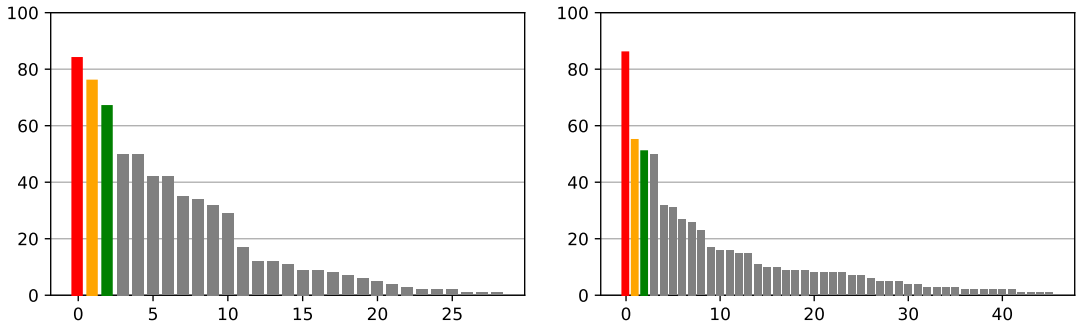
For example, the largest nodes in Fig. 5 (A) and (B) contain $49,388$ and $33,273$ data points, respectively, while the smallest nodes in both graphs contain only a single data point. The red color with the contour level of 1 indicates that the corresponding node is composed of all fraud data points and no fraud data point is contained if the level is 0. The mapper graph $\mathcal{G}_f$ contains a total of 60 nodes, with 14 consisting of more fraudulent than normal data, which we refer to as fraudulent nodes. These fraudulent nodes are divided into two subgraphs located on the left and right sides of the entire graph. Notably, the subgraph on the right is distinctly separated from the normal nodes. In contrast, in the mapper graph $\mathcal{G}_g$, there are 36 fraudulent nodes out of a total of 88 nodes. These fraudulent nodes form four subgraphs, with the subgraph in the bottom right exhibiting particularly high connectivity.

Seemingly, the fraudulent nodes are more clearly segregated from the normal nodes in the mapper graph $\mathcal{G}_g$ compared to $\mathcal{G}_f$. We utilize the node prediction algorithm, developed in Section 3, on test data to numerically compare the representation capabilities of the two graphs. Firstly, we observe that among $227,846$ test samples, the mapper graphs $\mathcal{G}_f$ and $\mathcal{G}_g$ fail to assign nodes to 34 and 28 samples, respectively. It is intriguing that $\mathcal{G}_f$ and $\mathcal{G}_g$ can encompass almost all unseen samples even though the mapper graphs were constructed with just the first 20% of data. The mapper graph $\mathcal{G}_g$ covers six more samples compared to $\mathcal{G}_f$. We speculate that the projection filter function has less diversity among samples. In contrast, the filter function f, defined as the norm and the anomaly scores of data, exhibits greater diversity across samples.

Figure 6 shows the distributions of the fraudulent samples in the test data over the nodes in the mapper graphs $\mathcal{G}_f$ and $\mathcal{G}_g$. On the x-axis, the nodes in each graph are arranged in order of the number of samples mapped into those nodes, which are shown on the y-axis. The three colored nodes correspond to the nodes circled in the same colors in Figure 5. Notably, the first two nodes (colored red and orange) are identified as fraudulent nodes in $\mathcal{G}_f$, but as normal nodes in $\mathcal{G}_g$. This indicates that a variety of samples are assigned to normal nodes in $\mathcal{G}_g$, which implies $\mathcal{G}_g$ may not represent future fraudulent data as effectively as $\mathcal{G}_f$. Hence, the mapper graph $\mathcal{G}_f$ more accurately preserves the overall underlying structure of the fraud detection data.

4.4. **Remarks on the node prediction algorithm.** The following remarks outline some characteristics of the prediction algorithm.

• The node prediction algorithm relies on the selection of filter functions used. As shown in the numerical results above, certain nodes are predicted by both $\mathcal{G}_f$ and $\mathcal{G}_g$, while some nodes are exclusively predicted by only one of the graphs. Similar considerations are applicable when the prediction algorithm returns the value of -1, indicating no identified nodes. For example, $\mathcal{G}_f$ and $\mathcal{G}_f$ failed to identify the associated nodes for 34 and 28 samples, respectively. Among them, there are 5 shared samples for which neither $\mathcal{G}_f$ nor $\mathcal{G}_f$ yielded associated nodes. In Fig. 6, we marked the nodes in each mapper graph, where fraudulent data are most prominent with circles in the graph. For $\mathcal{G}_f$, the top 2 nodes were fraudulent nodes, whereas for $\mathcal{G}_g$, the top 2 nodes were normal nodes. This indicates that the prediction algorithm depends on the choice of the filter function and there is no guarantee that newly injected fraud data will always be mapped solely to fraud nodes in the above example.



(A) Mapper graph $\mathcal{G}_f$ (B) Mapper graph $\mathcal{G}_g$

FIGURE 6. Distributions of the fraudulent data in the test dataset over the nodes in the mapper graphs $\mathcal{G}_f$ and $\mathcal{G}_g$. The x-axis indicates the nodes sorted by the number of fraudulent data that are mapped into the nodes, which are shown on the y-axis. The colors of the first three bars correspond to those used in Figure 5.

• The node identified by the prediction algorithm might contain multiple data points within it. Furthermore, if the prediction algorithm predicts multiple nodes for the input data, it implies a larger amount of data could be related to the input data. In such cases, statistical analysis becomes feasible.

• The prediction algorithm might yield predictive results in two potential cases. The first case involves the algorithm returning the node numbers associated with the input data – the identified nodes are the nodes within the mapper graph used by the prediction algorithm. In the second case, our algorithm returns a value of -1 because none of the nodes from the employed mapper graph is identified.

• Understanding the situation is important when none of the nodes from the utilized mapper graph is identified. This scenario could be explained by considering two possible reasons. Firstly, the absence of node identification might stem from insufficient data samples proximal to the input data. Alternatively, it could indicate that the input data inherently does not belong to the dataset forming the mapper graph.

5. CONCLUSION

The mapper algorithm, a TDA method, visualizes the given dataset as a graph. The key characteristics of the mapper algorithm is embedded well in the graph structure such as the node connectivity and clustering. As the data is represented as a graph, the overall shape of the graph delivers useful knowledge about the given data. Particularly for the detection problems, one can detect the desired feature characteristics of the new input data point by identifying the distinct sub-graphs or nodes where the new input data point belongs. Most mapper packages provide descriptive algorithms resulting in the static mapper graph and its overall shape. In

this paper, we developed an algorithm that can identify the sub-graphs or nodes where the new input data belongs. Once such information is extracted, one can determine whether the new input point falls into the category of interest. For example, if the category is the degree of anomaly, the method can be used for the anomaly detection. Moreover, as the computations for this method involve only the computation of the function values, covers and clusters, the computational complexity of the proposed algorithm is low. Thus this can be used as a real-time detection method. In this paper, we used the real credit card transaction data and considered the fraud transaction detection problem. The numerical experiments show that the developed algorithm efficiently identifies the node information associated with the new input data point. Our future research focuses on utilizing analysis methods associated with the node prediction algorithm to determine the characteristics of the input data.

ACKNOWLEDGMENTS

This work is supported by National Research Foundation of Korea under the grant number 2021R1A2C3009648 and POSTECH Basic Science Research Institute under the NRF grant number NRF2021R1A6A1A1004294412.

REFERENCES

[1] G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46:255—-308, 2009.
[2] F. Hensel, M. Moor, and B. Rieck. A survey of topological machine learning methods. *Frontiers Artificial Intelligence*, 4, 681108, 2021.
[3] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete Computational Geometry*, 37:103–120, 2007.
[4] A. Zomorodian and G. Carlsson. Computing persistent homology. *In: SCG '04 Proceedings of the Twentieth Annual Symposium on Computational Geometry*, page 347–356, 2004.
[5] C. Leesten and J.-H. Jung. Detection of gravitational waves using topological data analysis and convolutional neural network: An improved approach. *arXiv:1910.08245*, 2019.
[6] Jose A. Perea and John Harer. Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, 15(3):799–838, Jun 2015.
[7] Keunsu Kim and Jae-Hun Jung. Exact multi-parameter persistent homology of time-series data: Fast and variable one-dimensional reduction of multi-parameter persistence theory, 2023.
[8] John Nicponski and Jae-Hun Jung. Topological data analysis of vascular disease: A theoretical framework. *Frontiers in Applied Mathematics and Statistics*, 6, 2020.
[9] Gurjeet Singh, Facundo Memoli, and Gunnar Carlsson. Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. In M. Botsch, R. Pajarola, B. Chen, and M. Zwicker, editors, *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2007.
[10] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal M. Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration. *Journal of Machine Learning Research*, 22(39):1–6, 2021.
[11] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 1*, 2:559–572, 1901.
[12] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.
[13] Enrique Alvarado, Robin Belton, Emily Fischer, Kang-Ju Lee, Sourabh Palande, Sarah Percival, and Emilie Purvine. g-mapper: Learning a cover in the mapper construction, arXiv:2309.06634, 2023.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[15] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The gudhi library: Simplicial complexes and persistent homology. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014*, pages 167–174, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[16] Hendrik Jacob van Veen, Nathaniel Saul, David Eargle, and Sam W. Mangham. Kepler mapper: A flexible python implementation of the mapper algorithm. *Journal of Open Source Software*, 4(42):1315, 2019.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

APPENDIX A

The script below draws circles around the nodes to which $\mathbf{x}_1$ and $\mathbf{x}_2$ are assigned in the annulus example. The full script is available at `https://github.com/HiddenBeginner/mapper_prediction/blob/master/Examples.ipynb`

```
import plotly.graph_objects as go

nodelist = data2nodes(data, pipe, graph)
fig = plot_static_graph(graph, X, color_data=X[:,0], node_scale=20)

indices = nodelist[0]
fig.add_trace(
    go.Scatter(
        mode='markers',
        x=fig.data[1].x[indices],
        y=fig.data[1].y[indices],
        marker=dict(
            size=np.array(fig.data[1].marker.size)[indices] / 10,
            color='rgba(0, 0, 0, 0.0)',
            line=dict(color='red', width=5)
        ),
        showlegend=False
    )
)

indices = nodelist[1]
fig.add_trace(
    go.Scatter(
        mode='markers',
        x=fig.data[1].x[indices],
        y=fig.data[1].y[indices],
        marker=dict(
            size=np.array(fig.data[1].marker.size)[indices] / 10,
            color='rgba(0, 0, 0, 0.0)',
            line=dict(color='blueviolet', width=5)
        ),
        showlegend=False
    )
)

fig.show()
```

APPENDIX B

The following histograms show the distribution of individual feature values from V1 to V28. The x-axes indicate the feature values and the y-axes are the relative frequencies. These diagrams illustrate which features exhibit greater sensitivity towards fraud transactions.

FIGURE 7. The histograms of features from `V1` to `V28` within the training dataset of the fraud detection dataset. The x-axes indicate the feature values and the y-axes are the relative frequencies.